**Basic exercises (After You have completed ROS Tutorials)**

- For these exercises it is best to keep using the same type of camera (Xtion, Kinect 360, or Kinect2).
- Make sure you have created a git branch for your group. Make sure to put all your code and other files on the branch cause this makes it easier for us to check your code (Don't add your report on the repo though). Make sure you have added, commited, and pushed all files before sending in your report.
- Make sure you add your branch name in the report (branch names are capital sensitive).
- For each exercise make sure you include the name of the package you created or used (All packages should be created in the catkin_ws/src folder, full path on lab pc: /home/student/sudo/ros/catkin_ws/src ) and clear run instructions for all the nodes that are part of the exercise, including how to run the camera. You can assume we will use a single catkin_make command before running any code. Also include the type of camera you used.

**Exercise 1.**

In this exercise you need to create a single launch file that will create a transform between base_link and camera_link (or kinect2_link if you are using the kinect2).

Setup the tripod with camera and aim the camera to the floor with an angle of about 45 degrees.

Launch the camera and open rviz (rosrun rviz rviz). Set the fixed frame to "camera_link" (or "kinect2_link").
Add a PointCloud2 and a TF display.
For the PointCloud2 display you need to select to which topic the PointCloud2 needs to subscribe, select one (For the kinect2 only select the sd one, the other ones will slow down your pc a lot) and you will see a point cloud. As you can see the cloud does not line up with the grid. The goal of this exercise is to line up the floor with the grid, using a transformation.

Create a launch file that will run a static transform publisher from base_link to camera_link (Hint use google, the answer is literally on the ros wiki site) and make the transform so that the ground floor is now on the same level as the base_link (on the grid). Switch the fixed frame to "base_link", this topic name is only available after you launched your static transform cause the base_link has not been created before. Make sure the floor is in front of the base_link, where the red x-axis is pointing to.
The TF display will help you to visualize how the two links are being transformed.

Explain all parameters in the launch file.

**Exercise 2.**

In this exercise you need to adjust the URDF model of Alice (Tutorials about URDF: http://wiki.ros.org/urdf/Tutorials, these can help you understand more about URDF). You will need to add an extra Xtion camera on top of Alice.
All the necessary files are found in the alice_description package, in the urdf folder.
The main building block is the 'complete_model.urdf.xacro' file. This file will use different xacro files to create one urdf file.
You need to create a new xacro file and make sure that the 'complete_model.urdf.xacro' will make your new file. Take a look at the 'complete_model_bottom_part.urdf.xacro' and 'alice_head.urdf.xacro' files, these files will contain everything you need to complete this assignment.

The goal is to add an extra Xtion on top of Alice, by creating a new bar which is 30cm in length, and will fit right on top of the 'middle_bar', the Xtion camera needs to be added on top of this new bar, and needs to be facing down with an angle of 45 degrees.
(Don't forget the gazebo parts or else the camera won't work in simulation)

To generate the urdf file run the command: rosrun xacro xacro complete_model.urdf.xacro > alice.urdf
This should not give any warnings or errors.
You can test your new model by launching the simulation (Which you will use more of in the Navigation exercises). You can launch the simulation with the command: roslaunch alice_gazebo alice_simulation.launch
Use rviz to check if your TF is correct, and also if the image or point cloud from the Xtion is looking in the right direction.

Explain in detail what you did.

(After you received feedback for this exercise you can remove the part you added in the complete_model.urdf.xacro file, and rerun the command to make alice.urdf, cause you won't need the new camera for any other exercises.)

**Exercise 3.**

In this exercise you need to create multiple nodes that will communicate with each other.

The implementation is very open, but make sure you explain your design decisions. You can do this exercise in c++ or python.

- Create a node that receives a RGB image from a camera (Xtion, Kinect 360, or Kinect2). Hint: Use “rostopic type XXX” for the image topic to understand the type of the image required for defining the callback function
- Create a service that will convert a RGB image into a grayscale image.
- Create an action server that requires a RGB image and a grayscale image for its Goal, of which it will then calculate the total amount of pixels in the two images combined (Hint: a RGB image consists of 3 channels). The result of the action server will be the total amount of pixels. (There is no need for sending feedback)
- Create a single subscriber node that will receive the RGB image, grayscale image, and the total amount of pixels as calculated by the action server. It will show each image (use OpenCV) in a window, with correct labeling, and will print the total amount of pixels to the console.

Connect all nodes together as you seem fit, but make sure none of the nodes gives any warning messages. For both the action server and the final subscriber node, make sure you only use one callback function in each node. Note that the camera will keep sending images, so your final node should also be kept updated.
Below is python code for the conversion between the ros image type (sensor_msgs/Image) and OpenCV (Numpy).

Python Conversion Code:

```
#!/usr/bin/env python
import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
import cv2
import numpy

## convert from sensor_msg format to opencv format (Numpy)
def convert_image_cv(data, type = "bgr8"): # use type = "8UC1" for grayscale images
        try:
        cv_image = bridge.imgmsg_to_cv2(data, type) # use "bgr8" if its a color image
        except CvBridgeError, e:
        print e
```

```
        ### Show the image
#       cv_image = numpy.asarray(cv_image)
#       cv2.imshow("Camera Image", cv_image)
#       cv2.waitKey(1)

        return cv_image

## Convert from opencv format to sensor_msgs format
def convert_image_ros(data, type = "bgr8"): # use type = "8UC1" for grayscale images
        try:
        ros_image = bridge.cv2_to_imgmsg(data, type)
        except CvBridgeError, e:
        print e

        return ros_image

if __name__ == '__main__':
        bridge = CvBridge()
```